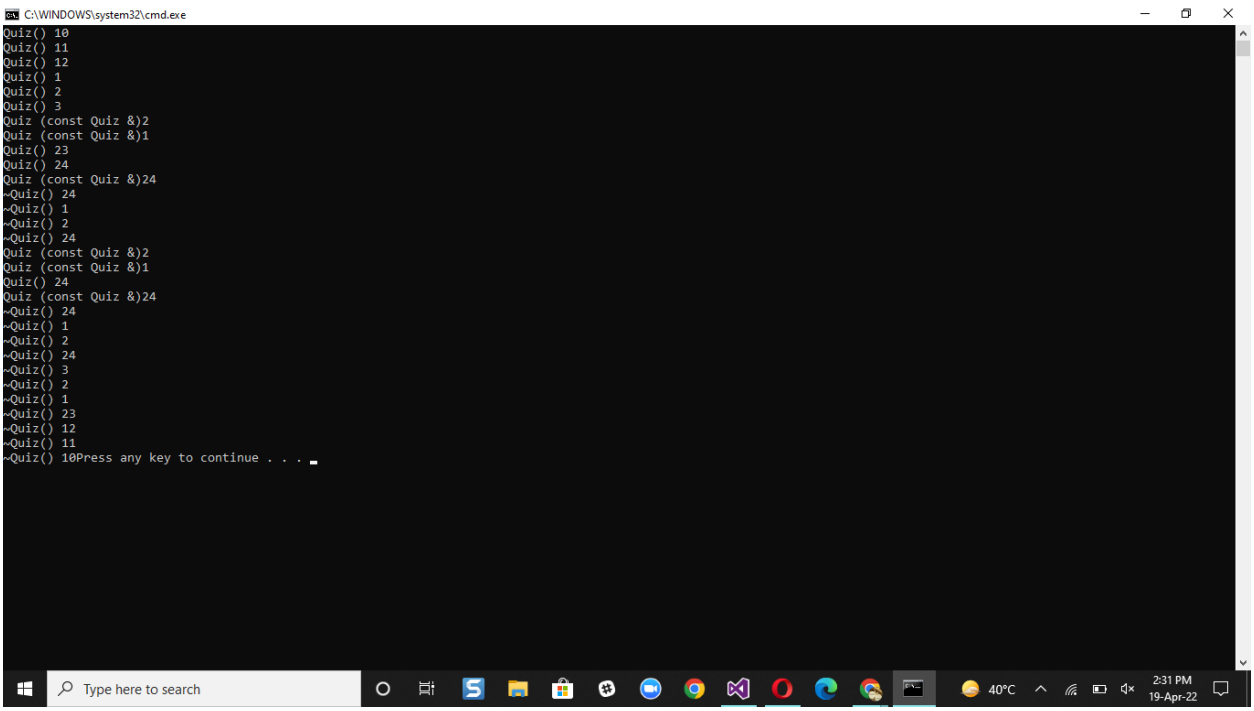
```cpp
#include <iostream>
using namespace std;
class Quiz {
public:
 int a;
 Quiz(int i = 1)
 {
 a = i;
 cout << "\nQuiz() " << a;
 }
 Quiz(const Quiz& ref)
 {
 a = ref.a;
 cout << "\nQuiz (const Quiz &)"
 << a;
 }
 ~Quiz()
 {
 cout << "\n~Quiz() " << a;
 }
};
Quiz f(Quiz a, Quiz b, Quiz& c)
{
 static Quiz y(23);
 Quiz x(24);
 return x;
}
Quiz g1(10);
static Quiz g2(11);
Quiz g3(12);
int main()
{
 Quiz r, s(2), t(3);
 f(r, s, t);
 f(r, s, t);
}
```



Output:

Quiz() 10

Quiz() 11

Quiz() 12

Quiz() 1

Quiz() 2

Quiz() 3

Quiz (const Quiz &)2

Quiz (const Quiz &)1

Quiz() 23

Quiz() 24

Quiz (const Quiz &)24

~Quiz() 24

~Quiz() 1

~Quiz() 2

~Quiz() 24

Quiz (const Quiz &)2

Quiz (const Quiz &)1

Quiz() 24

Quiz (const Quiz &)24

~Quiz() 24

~Quiz() 1

~Quiz() 2

~Quiz() 24

~Quiz() 3

~Quiz() 2

~Quiz() 1

~Quiz() 23

~Quiz() 12

~Quiz() 11

~Quiz() 10

Explanation:

- In this question, you just had to see which object would be created first and which would be created later. And we know, the calling sequence of destructors is the reverse of calling sequence of constructors.

- There are some global and static objects in the program, which are written before main.

Quiz g1(10);

static Quiz g2(11);

Quiz g3(12);

- So, when program starts, g1 will be created, leaving the output “Quiz() 10”

- After that g2 will be created, leaving the output “Quiz() 11”

- Now, it’s the turn of g3, leaving the output “Quiz() 12”

- Now, main function will start executing.

- Here, objects r, s and t are constructed with values 1, 2 and 3 respectively. So the outputs are “Quiz() 1”, “Quiz() 2” and “Quiz() 3”.

- Now, function f is called with parameters r, s, and t. The first two parameters are passed by value, and the third is passed by reference, so copy constructor will be called for the first two parameters. But the copy of parameters is always created in reverse order, i.e., the copy of second parameter will be created first, and then the copy of first parameter will be created. So, leaving the outputs “Quiz (const Quiz &)2” and “Quiz (const Quiz &)1”.

- Inside the f function, a static object y is created, which leaves the output “Quiz() 23”.

- Now, it’s the turn of object x. Which leaves the output “Quiz() 24”.

- As, x is being returned by value, so a copy of x would be made to return, which leaves the output “Quiz (const Quiz &)24”.

- Now, the function is about to end, so all local objects would be destroyed. Local objects are a, b, and x (y is static, so it won’t be destroyed now). The local objects were constructed in the order b, a, and x. So the reverse order is x, a, b. So the outputs “~Quiz() 24”, “~Quiz() 1”, and “~Quiz 2”.

- Now, when we return to the main, a copy of x was returned from f, but it isn’t stored anywhere. Which will be destroyed now. So, “~Quiz() 24”.

- Now, there is another call to f with the same parameters. The order of constructor calling, and destructor calling would be the same with one exception that this time, the static object y won’t be created, as it has already been created. So, the outputs will be “Quiz (const Quiz &)2”, “Quiz (const Quiz &)1”, “Quiz() 24”, “Quiz (const Quiz &)24”, “~Quiz() 24”, “~Quiz() 1”, “~Quiz() 2”, “~Quiz() 24”.

- Now, after the second call to f, in main, the local objects r, s, and t would be destroyed, whose order of destruction would be t, s, r. So outputs “~Quiz() 3”, “~Quiz() 2”, “~Quiz() 1”.

- Now, the program is about to end, and the global and static objects g1, g2, g3, and y are still left to destruct. Their order of creation was g1, g2, g3, y, so order of destruction would be y, g3, g2, g1. So the outputs “~Quiz() 23”, “~Quiz() 12”, “~Quiz() 11”, “~Quiz() 10”.